

## ▼ Basic Image Enhancement Using Mathematical Operations

Image Processing techniques take advantage of mathematical operations to achieve different results. Most often we arrive at an enhanced version of the image using some basic operations. We will take a look at some of the fundamental operations often used in computer vision pipelines. In this notebook we will cover:

- Arithmetic Operations like addition, multiplication
- Bitwise Operations like OR, AND, XOR, NOT

```
import os
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
```

### ▼ Original image

```
img_bgr = cv.imread("Images/messi5.jpg", cv.IMREAD_COLOR)
img_rgb = cv.cvtColor(img_bgr, cv.COLOR_BGR2RGB)
```

```
# Display 18x18 pixel image.
plt.imshow(img_rgb)
```

```
<matplotlib.image.AxesImage at 0x1333c97d150>
```



## ▼ Addition or Brightness

The first operation we discuss is simple addition of images. This results in increasing or decreasing the brightness of the image since we are eventually increasing or decreasing the intensity values of each pixel by the same amount. So, this will result in a global increase/decrease in brightness.

```
matrix = np.ones(img_rgb.shape, dtype="uint8") * 50

img_rgb_brighter = cv.add(img_rgb, matrix)
img_rgb_darker = cv.subtract(img_rgb, matrix)

# Show the images
plt.figure(figsize=[18, 5])
plt.subplot(131); plt.imshow(img_rgb_darker); plt.title("Darker");
plt.subplot(132); plt.imshow(img_rgb); plt.title("Original");
plt.subplot(133); plt.imshow(img_rgb_brighter); plt.title("Brighter");
```



## ▼ Multiplication or Contrast

Just like addition can result in brightness change, multiplication can be used to improve the contrast of the image.

Contrast is the difference in the intensity values of the pixels of an image. Multiplying the intensity values with a constant can make the difference larger or smaller ( if multiplying factor is  $< 1$  ).

```
matrix1 = np.ones(img_rgb.shape) * 0.8
matrix2 = np.ones(img_rgb.shape) * 1.2

img_rgb_darker = np.uint8(cv.multiply(np.float64(img_rgb), matrix1))
img_rgb_brighter = np.uint8(cv.multiply(np.float64(img_rgb), matrix2))

# Show the images
plt.figure(figsize=[18,5])
plt.subplot(131); plt.imshow(img_rgb_darker); plt.title("Lower Contrast");
plt.subplot(132); plt.imshow(img_rgb); plt.title("Original");
plt.subplot(133); plt.imshow(img_rgb_brighter); plt.title("Higher Contrast");
```



## What happened?

Can you see the weird colors in some areas of the image after multiplication?

The issue is that after multiplying, the values which are already high, are becoming greater than 255. Thus, the overflow issue. How do we overcome this?

## ▼ Handling Overflow using np.clip

```
matrix1 = np.ones(img_rgb.shape) * 0.8
matrix2 = np.ones(img_rgb.shape) * 1.2

img_rgb_lower = np.uint8(cv.multiply(np.float64(img_rgb), matrix1))
img_rgb_higher = np.uint8(np.clip(cv.multiply(np.float64(img_rgb), matrix2), 0, 255))

# Show the images
plt.figure(figsize=[18,5])
```

```
plt.subplot(131); plt.imshow(img_rgb_lower); plt.title("Lower Contrast");
plt.subplot(132); plt.imshow(img_rgb); plt.title("Original");
plt.subplot(133); plt.imshow(img_rgb_higher); plt.title("Higher Contrast");
```



## ▼ Bitwise Operations

### Function Syntax

Example API for `cv.bitwise_and()`. Others include: `cv.bitwise_or()`, `cv.bitwise_xor()`, `cv.bitwise_not()`

```
dst = cv.bitwise_and( src1, src2[, dst[, mask]] )
```

`dst`: Output array that has the same size and type as the input arrays.

The function has **2 required arguments**:

1. `src1`: first input array or a scalar.
2. `src2`: second input array or a scalar.

An important optional argument is:

1. `mask`: optional operation mask, 8-bit single channel array, that specifies elements of the output array to be changed.

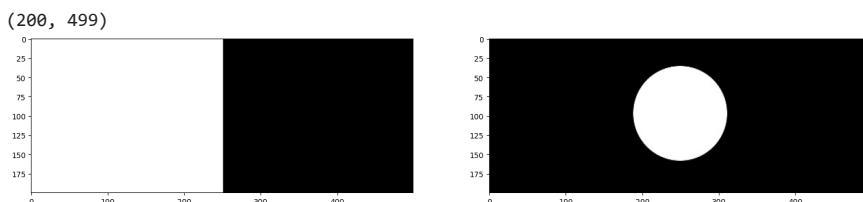
### OpenCV Documentation

1. [Arithmetic Operations on Images](#)
2. [bitwise\\_and\(...\)](#) function

```
img_rec = cv.imread("Images/rectangle.jpg", cv.IMREAD_GRAYSCALE)
```

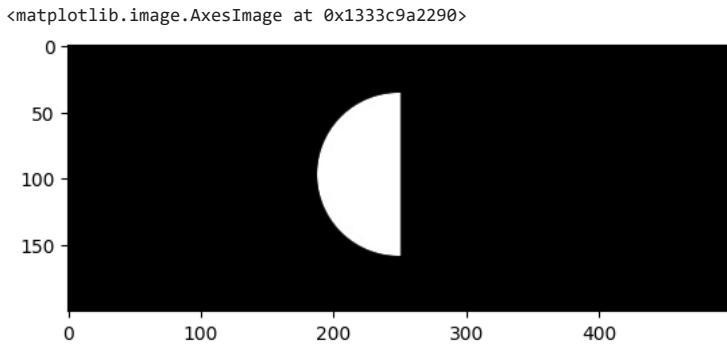
```
img_cir = cv.imread("Images/circle.jpg", cv.IMREAD_GRAYSCALE)
```

```
plt.figure(figsize=[20, 5])
plt.subplot(121); plt.imshow(img_rec, cmap="gray")
plt.subplot(122); plt.imshow(img_cir, cmap="gray")
print(img_rec.shape)
```



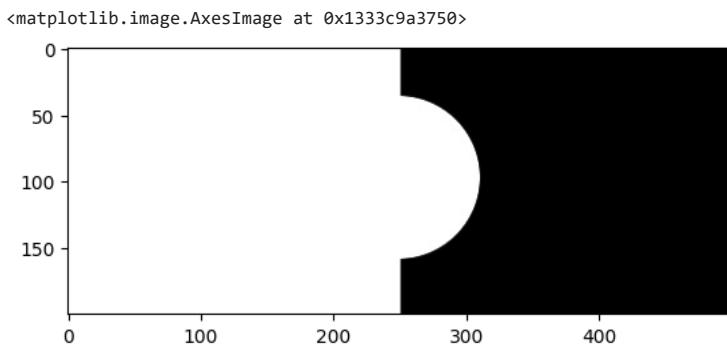
## ▼ Bitwise AND Operator

```
result = cv.bitwise_and(img_rec, img_cir, mask=None)
plt.imshow(result, cmap="gray")
```



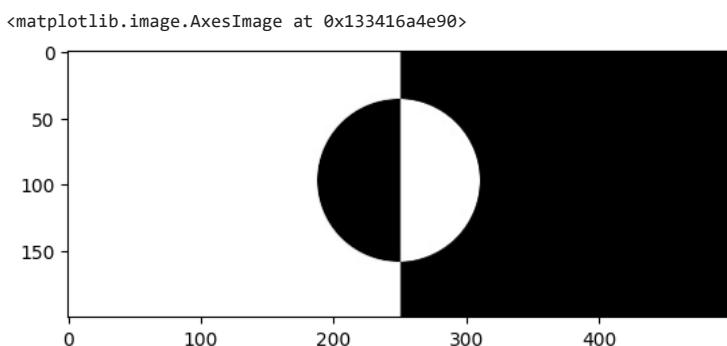
#### ▼ Bitwise OR Operator

```
result = cv.bitwise_or(img_rec, img_cir, mask=None)
plt.imshow(result, cmap="gray")
```



#### ▼ Bitwise XOR Operator

```
result = cv.bitwise_xor(img_rec, img_cir, mask=None)
plt.imshow(result, cmap="gray")
```

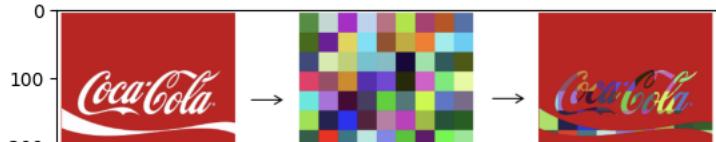


#### ▼ Application: Logo Manipulation

In this section we will show you how to fill in the white lettering of the Coca-Cola logo below with a background image.

```
image=cv.imread("Images/Logo_Manipulation.png")
image=cv.cvtColor(image, cv.COLOR_BGR2RGB)
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x13341680350>
```



#### ▼ Read Foreground image

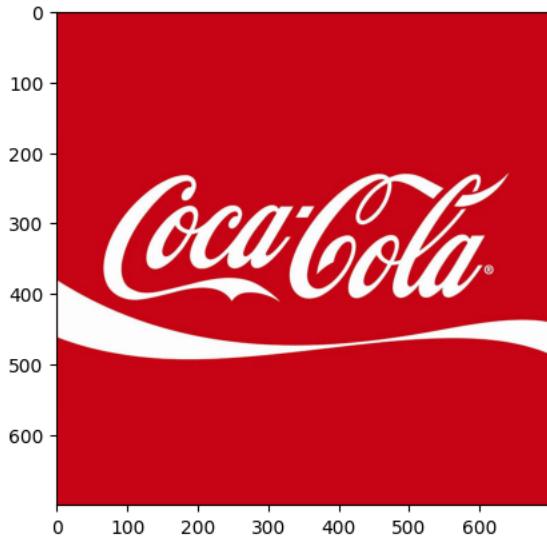
```
0      200      400      600      800
```

```
img_bgr = cv.imread("Images/coca-cola-logo.png")
img_rgb = cv.cvtColor(img_bgr, cv.COLOR_BGR2RGB)
plt.imshow(img_rgb)
```

```
print(img_rgb.shape)
```

```
logo_w = img_rgb.shape[0]
logo_h = img_rgb.shape[1]
```

```
(700, 700, 3)
```



#### ▼ Read Background image

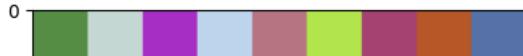
```
# Read in image of color cheackerboard background
img_background_bgr = cv.imread("Images/checkerboard_color.png")
img_background_rgb = cv.cvtColor(img_background_bgr, cv.COLOR_BGR2RGB)
```

```
# Set desired width (logo_w) and maintain image aspect ratio
aspect_ratio = logo_w / img_background_rgb.shape[1]
dim = (logo_w, int(img_background_rgb.shape[0] * aspect_ratio))
```

```
# Resize background image to same size as logo image
img_background_rgb = cv.resize(img_background_rgb, dim, interpolation=cv.INTER_AREA)
```

```
plt.imshow(img_background_rgb)
print(img_background_rgb.shape)
```

```
(700, 700, 3)
```



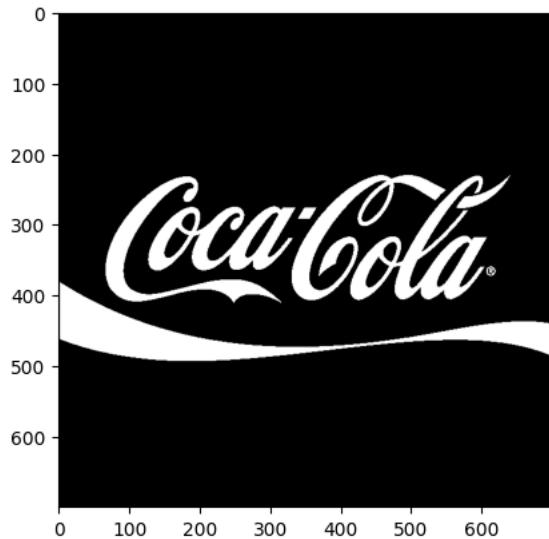
#### ▼ Create Mask for original Image

```
img_gray = cv.cvtColor(img_rgb, cv.COLOR_RGB2GRAY)

# Apply global thresholding to create a binary mask of the logo
retval, img_mask = cv.threshold(img_gray, 127, 255, cv.THRESH_BINARY)

plt.imshow(img_mask, cmap="gray")
print(img_mask.shape)
```

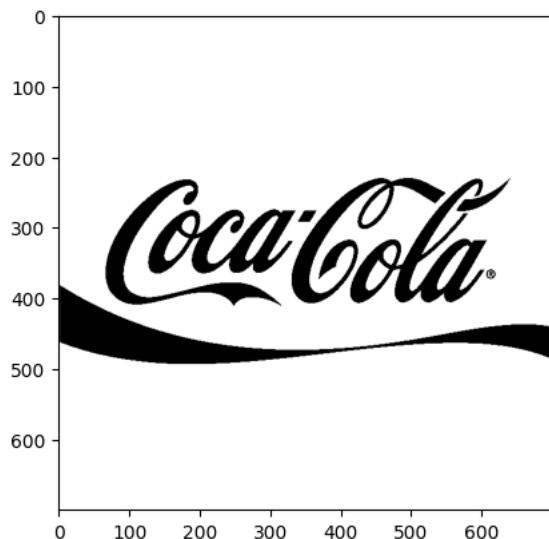
```
(700, 700)
```



#### ▼ Invert the Mask

```
# Create an inverse mask
img_mask_inv = cv.bitwise_not(img_mask)
plt.imshow(img_mask_inv, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x1334003b750>
```

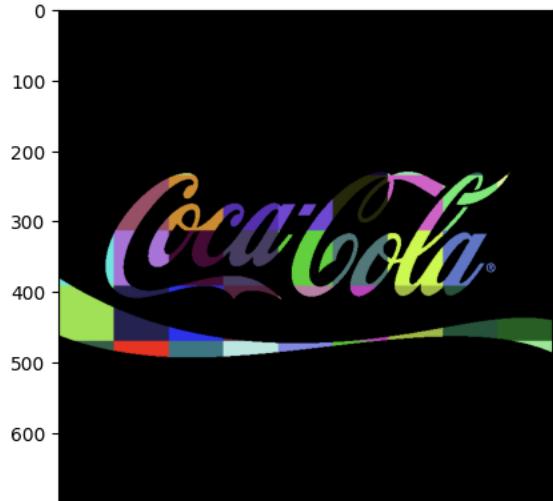


#### ▼ Apply background on the Mask

```
# Create colorful background "behind" the logo lettering
img_background = cv.bitwise_and(img_background_rgb, img_background_rgb, mask=img_mask)

plt.imshow(img_background)
```

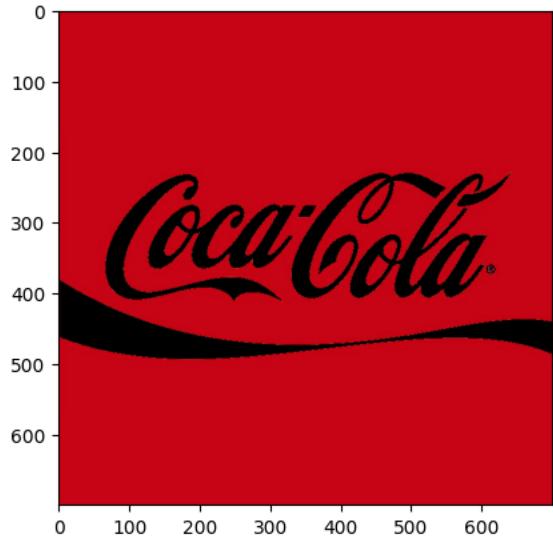
```
<matplotlib.image.AxesImage at 0x1333c5d0350>
```



#### ▼ Isolate foreground from image

```
# Isolate foreground (red from original image) using the inverse mask
img_foreground = cv.bitwise_and(img_rgb, img_rgb, mask=img_mask_inv)
plt.imshow(img_foreground)
```

```
<matplotlib.image.AxesImage at 0x133406699d0>
```



#### ▼ Result: Merge Foreground and Background

```
# Add the two previous results obtain the final result
result = cv.add(img_background, img_foreground)
plt.imshow(result)
cv.imwrite("logo_final.png", result[:, :, ::-1])
```



Image Negation

```
200 +-----+
```

```
img = cv.imread('Images/messi5.jpg')
# Show the image
plt.imshow(img), plt.title('Orginal Color Image')
img=img[:, :, ::-1]
```

Orginal Color Image



```
plt.imshow(img), plt.title('Orginal Color Image')
```

```
(<matplotlib.image.AxesImage at 0x13340734090>,
Text(0.5, 1.0, 'Orginal Color Image'))
```

Orginal Color Image

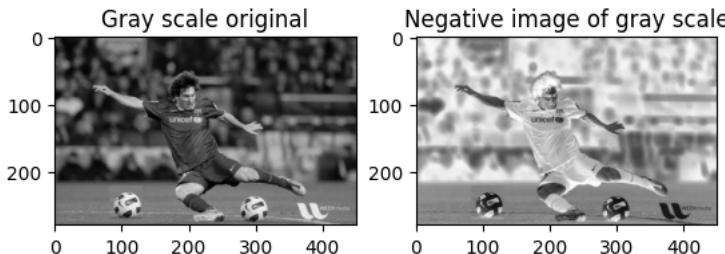


```
img_neg = cv.bitwise_not(img)
# Show the image
plt.imshow(img_neg), plt.title('color negative')
```

```
(<matplotlib.image.AxesImage at 0x133406b8350>,
Text(0.5, 1.0, 'color negative'))
```

color negative

```
img = cv.imread('Images/messi5.jpg',0)
img=img[::-1]# Invert the image using cv.bitwise_not
img_neg = cv.bitwise_not(img)
# Show the image
plt.subplot(121),plt.imshow(img,cmap="gray"),plt.title('Gray scale original')
plt.subplot(122),plt.imshow(img_neg,cmap="gray"),plt.title('Negative image of gray scale')
plt.show()
```



## ▼ Goal

Learn several arithmetic operations on images, like addition, subtraction, bitwise operations, and etc. Learn these functions: cv.add(), cv.addWeighted(), etc.

### Image Addition

You can add two images with the OpenCV function, cv.add(), or simply by the numpy operation `res = img1 + img2`.

Both images should be of same depth and type, or the second image can just be a scalar value.

### Note

There is a difference between OpenCV addition and Numpy addition. OpenCV addition is a saturated operation while Numpy addition is a modulo operation.

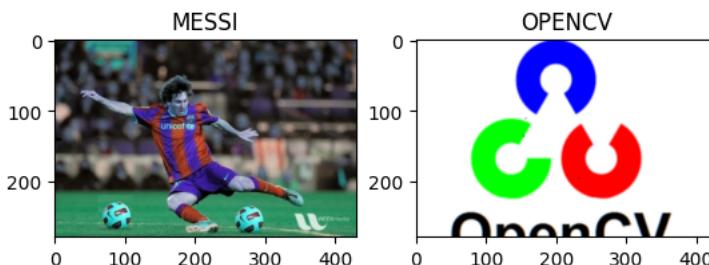
```
x = np.uint8([250])
y = np.uint8([10])
print(cv.add(x,y)) # 250+10=> 255 saturation operation
print(x+y) # modulo operation, 250,8 bits can store upto 255, 0,1,2,3,4 so value is 4

[[255]]
[4]
```

Double-click (or enter) to edit

```
img1 = cv.imread('Images/messi5.png')
img2 = cv.imread('Images/opencv-logo.png')
assert img1 is not None, "file could not be read, check with os.path.exists()"
assert img2 is not None, "file could not be read, check with os.path.exists()

plt.subplot(121),plt.imshow(img1),plt.title('MESSI')
plt.subplot(122),plt.imshow(img2),plt.title('OPENCV')
plt.show()
```



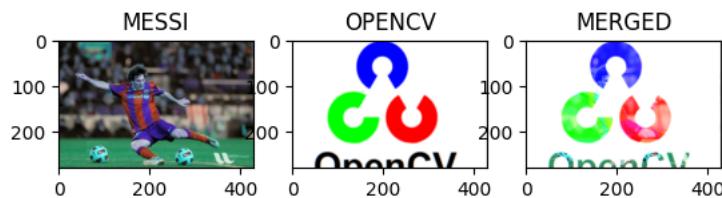
```
#To add, two image sizes should be same
print("To add, two image sizes should be same")
```

```
print(img1.shape)
print(img2.shape)
```

To add, two image sizes should be same  
(280, 430, 3)  
(280, 430, 3)

```
dst=cv.add(img1, img2)
plt.subplot(131),plt.imshow(img1),plt.title('MESSI')
plt.subplot(132),plt.imshow(img2),plt.title('OPENCV')
plt.subplot(133),plt.imshow(dst),plt.title('MERGED')
```

```
plt.show()
```



#### ▼ Image Blending

This is also image addition, but different weights are given to images in order to give a feeling of blending or transparency. Images are added as per the equation below:

$$g(x) = (1-\alpha)f_0(x) + \alpha f_1(x)$$

By varying  $\alpha$  from 0→1, you can perform a cool transition between one image to another.

Here I took two images to blend together. The first image is given a weight of 0.7 and the second image is given 0.3.

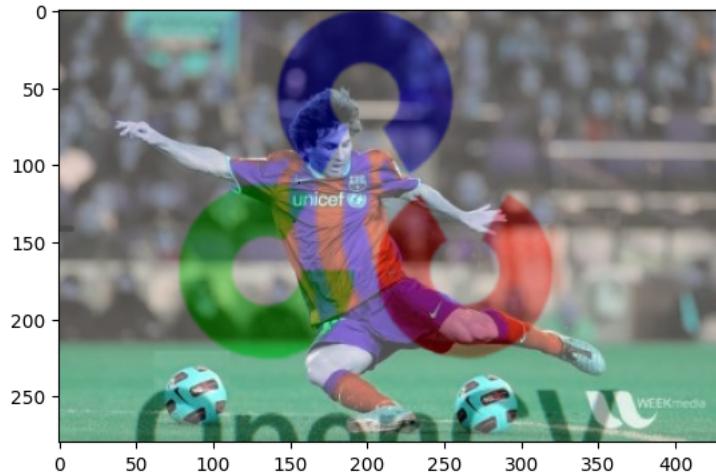
`cv.addWeighted()` applies the following equation to the image:

$$dst = \alpha \cdot img1 + \beta \cdot img2 + \gamma$$

Here  $\gamma$  is taken as zero.

```
dst = cv.addWeighted(img1,0.7,img2,0.3,0)
plt.imshow(dst)
```

```
<matplotlib.image.AxesImage at 0x13340719150>
```



Double-click (or enter) to edit

```
dst = cv.addWeighted(img1,0.2,img2,0.8,0)
plt.subplot(131),plt.imshow(img1),plt.title('MESSI')
plt.subplot(132),plt.imshow(img2),plt.title('OPENCV')
plt.subplot(133),plt.imshow(dst),plt.title('MERGED')
```

```
plt.show()
```

